Efficient Document-at-a-Time and Score-at-a-Time Query Evaluation for Learned Sparse Representations

JOEL MACKENZIE, The University of Queensland, Australia
ANDREW TROTMAN, University of Otago, New Zealand
JIMMY LIN, University of Waterloo, Canada

Researchers have had much recent success with ranking models based on so-called learned sparse representations generated by transformers. One crucial advantage of this approach is that such models can exploit inverted indexes for top-k retrieval, thereby leveraging decades of work on efficient query evaluation. Yet, there remain many open questions about how these learned representations fit within the existing literature, which our work aims to tackle using four representative learned sparse models. We find that impact weights generated by transformers appear to greatly reduce opportunities for skipping and early exiting optimizations in well-studied document-at-a-time (DaaT) approaches. Similarly, "off-the-shelf" application of score-at-a-time (SaaT) processing exhibits a mismatch between these weights and assumptions behind accumulator management strategies. Building on these observations, we present solutions to address deficiencies with both DaaT and SaaT approaches, yielding substantial speedups in query evaluation. Our detailed empirical analysis demonstrates that both methods lie on the effectiveness–efficiency Pareto frontier, indicating that the optimal choice for deployment depends on operational constraints.

CCS Concepts: • **Information systems** → **Search engine architectures and scalability**; **Search engine indexing**; **Information retrieval query processing**; **Retrieval efficiency**.

Additional Key Words and Phrases: Efficiency; Indexing; Query Processing; Learned Sparse Retrieval

ACM Reference Format:
Joel Mackenzie, Andrew Trotman, and Jimmy Lin. 2022. Efficient Document-at-a-Time and Score-at-a-Time Query Evaluation for Learned Sparse Representations. *ACM Transactions on Information Systems* 1, 1, Article 1 (January 2022), 29 pages. https://doi.org/10.1145/3576922

1 INTRODUCTION

Despite various investigations of alternatives over the past few decades, document-at-a-time (DaaT) query evaluation algorithms remain the dominant solution for the top-k retrieval problem that lies at the core of systems-focused information retrieval research as well as deployed production systems. The most recent systematic exploration of different query evaluation strategies that we are aware of is the work of Crane et al. [9], who compared the then latest score-at-a-time (SaaT) query evaluation algorithm against the WAND-family of DaaT index-traversal techniques [5, 18]. They concluded that despite advances in score-at-a-time query evaluation, the best approach at the time, known as JASS [37, 70], was still slower than Block-Max WAND (BMW) in terms of mean query latency, although the query latency of JASS was more consistent, with much lower tail latency [14].

Authors' addresses: Joel Mackenzie, The University of Queensland, Brisbane, Australia; Andrew Trotman, University of Otago, Dunedin, New Zealand; Jimmy Lin, University of Waterloo, Ontario, Canada.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1046-8188/2022/1-ART1 $15.00
https://doi.org/10.1145/3576922

Since the work of Crane et al. [9], however, the research landscape has changed in one important way: the introduction of a new class of retrieval models that rely on learned sparse representations. These models take advantage of transformer-based pretrained language models to assign term weights to bag-of-words representations that can be interpreted as impact scores, and thus top-*k* retrieval with these models can be accomplished using standard inverted indexes. Recent empirical studies have shown that this new class of models yields effectiveness that is superior to unsupervised retrieval models such as BM25, and at least on par with popular dense retrieval techniques [35]. Thus, it is important to study the behavior of query evaluation techniques—both DaaT as well as SaaT methods—in the context of these models.

Interestingly, we observe that the neural models that underlie these learned sparse retrieval models often assign counter-intuitive weights to terms in bag-of-words representations. These weights are counter-intuitive in at least two ways: First, the distribution of term weights does not appear to allow standard DaaT optimizations to effectively perform skipping and early exiting, which represent the source of accelerated query evaluation performance with traditional bag-of-words scoring models like BM25. Second, manual examination of the assigned term weights reveals that high scores are frequently assigned to terms in unexpected ways, for example, large weights placed on stopwords and subwords that lack any meaningful semantic content. These weight assignments are particularly puzzling since the models have been demonstrated to be more effective than existing bag-of-words models such as BM25 on various benchmark datasets.

From this starting point, we view our work as making three important contributions:

(1) We systematically characterize the differences between these learned sparse representations and their effects on query evaluation performance. Although these issues have been observed before [41, 55], our work confirms that term weights arising from learned models can negatively impact query performance.
(2) We are the first to compare DaaT and SaaT approaches in the context of learned sparse representations and show that these negative performance traits affect both DaaT and SaaT techniques, but in different ways. For DaaT approaches, the term weight distributions appear to greatly reduce opportunities for skipping, thus reducing the benefits of standard optimizations. These same weight distributions also force SaaT retrieval to use score accumulators in an inefficient manner, also slowing down the retrieval process.
(3) We introduce optimizations to address the deficiencies we observed in DaaT and SaaT algorithms above, leading to over 2× speedups without any appreciable loss in effectiveness.

On the whole, we demonstrate that both the accelerated DaaT and SaaT algorithms lie on the effectiveness/efficiency Pareto frontier across a variety of learned sparse models, meaning that the best choice of model and algorithm depends on operational constraints. The findings presented in this work establish a series of best practices for deploying learned sparse models in traditional inverted index architectures, and provide a reference for academics or practitioners wishing to experiment with learned sparse models in their own systems.

The remainder of this paper is organized as follows: Section 2 discusses background and related work, including efficient indexing and search. Section 3 outlines our experimental setup and evaluation framework. Section 4 details a preliminary experiment demonstrating the baseline efficiency of learned sparse models in both DaaT and SaaT retrieval systems. Section 5 analyzes the deficiencies of learned sparse models to gain a better understanding of when and why they are slower than traditional models. Section 6 then applies various optimization methods, both novel and existing, to DaaT and SaaT traversal to accelerate retrieval over learned sparse models. Finally, Section 8 concludes this research and discusses future work.

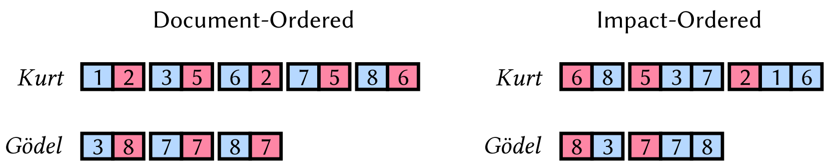

Fig. 1. Postings lists corresponding to the query *"Kurt Gödel"* with document-ordered postings shown on the left, and the equivalent impact-ordered postings shown on the right. Document identifiers are shown as blue squares and the corresponding impacts are shown as red squares. The document-ordered lists are sorted ascending on document identifiers, whereas the impact-ordered lists are sorted descending on the impacts and then sorted within each impact by ascending document identifier.

2 BACKGROUND AND RELATED WORK

The standard formulation of document retrieval using bag-of-words representations can be distilled into the following scoring function:

$$S_{d,q} = \sum_{t \in d \cap q} w_{d,t} \cdot w_{q,t} \tag{1}$$

where $w_{d,t}$ is the weight of term t in document d and $w_{q,t}$ represents the weight of term t in query q. Document weights are typically computed via a function of term statistics such as term frequencies, (inverse) document frequencies, document lengths, and so on. Query weights are often set to one (i.e., a multi-hot vector), which simplifies query–document scores to the sum of document weights of terms that are found in the query. This formulation covers nearly all major families of retrieval models (probabilistic [29, 65, 72], vector space, language modeling [63], divergence from randomness [1]) and is equivalent to the inner product of two weighted vectors of dimension $|V|$, where V is the vocabulary of the collection.

2.1 Efficient Indexing and Ranking

How to efficiently generate a top-k ranking of documents from an arbitrarily large collection $C = \{d_i\}_{i=0}^{N}$ in terms of $S_{d,q}$ has been, quite literally, the subject of many decades of intense study. In the IR context, this is often referred to as the query evaluation problem (or alternatively, the top-k retrieval/ranking problem), and nearly all modern algorithms exploit an inverted index where postings lists are systematically visited (i.e., traversed) in order to generate the top-k documents, stored in data structures such as min-heaps, accumulator tables, or a combination thereof. We now review some of the key data structures and algorithms used to facilitate efficient large-scale retrieval; we refer the interested reader to surveys by Zobel and Moffat [83] and Tonellotto et al. [69] for more detailed overviews.

Document-Ordered Indexes. Document-ordered indexes store, for each unique term t, a *postings list* containing a strictly increasing list of numeric document identifiers corresponding to documents that contain t. These document identifiers are assigned arbitrarily, often according to the order in which documents are indexed, although any ordering is possible (see below for additional discussion on document identifier reassignment). Alongside these identifiers is typically a secondary payload; this payload is often the number of times t appears in d (also known as the term frequency), but alternatives such as pre-computed impact values are also possible. Figure 1 (left) shows this layout. Based on best practices, document-ordered indexes compress fixed-sized *blocks* of identifiers interleaved with fixed-sized payload blocks, thereby allowing portions of individual postings lists to be selectively decoded.

Document-at-a-Time Retrieval. Given a document-ordered index, the most natural implementation of top-k retrieval follows the *document-at-a-time* (DaaT) strategy. At a high level, DaaT approaches work on postings lists that are increasing in document identifier and achieve low latency query evaluation by "avoiding work". That is, through the use of auxiliary data structures and per-term score upper-bounds, algorithms can cleverly work out when one or more documents cannot possibly be in the top-k and thereby "skip" them (often en masse). These optimized retrieval methods are commonly referred to as dynamic pruning algorithms.

Researchers have been exploring these algorithms since the 1980s. The simplest approaches include MaxScore [74] and WAND [5]. Both of these methods store, during indexing, the largest score contribution ($w_{d,t}$) observed across each postings list in the index; this term-wise upper-bound score, denoted U_t, can then be used to accelerate retrieval. In particular, the values of U_t across the query terms can be used to derive score estimates for a document *before* scoring it. Using a min-heap data structure to keep track of the top-k highest scoring documents at any given moment, these estimates can then be used to determine when documents *cannot* possibly enter the top-k set of interest, and this allows the algorithm to skip over them without actually computing the scores. This, in turn, can avoid unnecessary decoding and scoring operations.

Beyond WAND and MaxScore, a variety of more advanced dynamic pruning algorithms have been explored. Most of these advanced dynamic pruning mechanisms employ block-based upper-bound scores to generate better local upper-bound score estimates, trading off space (necessary to store auxiliary statistics to aid in the upper-bound computation) for accelerated retrieval [6, 18]. Presently, Variable Block-Max WAND (VBMW) [51] is generally acknowledged to represent the state of the art, although the best choice of DaaT traversal algorithm can depend on properties of the collection, ranker, queries, and specific optimizations enabled [31, 44, 53, 56, 60].

At an intuitive level, DaaT approaches can be informally characterized as accelerating query evaluation by "avoiding work", and the latest techniques accomplish this by "being smarter" to figure out what work can be avoided (e.g., which postings blocks to skip). However, this unavoidably requires storing additional term statistics in the inverted index and additional computations to reason over them. Naturally, the efficiency gains depend on the distribution of term weights that are generated by the underlying scoring functions. As we show in this work, it appears that the distribution of term weights generated by transformers in learned sparse representations are different from those expected by these techniques. Thus, dynamic pruning algorithms often end up spending a lot of computational effort without much efficiency gains.

Impact-Ordered Indexes. Impact-ordered indexes represent a completely different arrangement compared to document-ordered indexes. Instead of organizing postings lists based on the underlying document identifiers, impact-ordered indexes collect documents sharing the same impact score together, sorting these *segments* from high to low impact score. To this end, impact-ordered indexes rely on quantization, where the $w_{d,t}$ values are computed offline and quantized into integer values in some range $[0, 2^b - 1]$, with b representing the total number of bits required to store the highest impact value [8]. Within each segment, the document identifiers are sorted in a strictly increasing manner [2]. Typically, impact-ordered postings lists are compressed on a per-segment basis, allowing flexibility on which segments to process during retrieval. Figure 1 (right) shows this arrangement.

Score-at-a-Time Retrieval. Top-k retrieval over impact-ordered indexes is realized via *score-at-a-time* (SaaT) retrieval. Query evaluation proceeds by visiting each of the postings segments in decreasing order of the segment impact score. Scoring a single segment involves decoding the list of associated document identifiers, and for each document, adding the impact score to a corresponding *accumulator*. Typically, these accumulators are stored in a large table-like structure, with one accumulator allocated to each document (in the corpus), and various optimizations for efficiently managing these structures have been explored [28, 45].

Instead of "avoiding work", the intuition behind SaaT is to perform the necessary work as efficiently as possible. Operationally, this means organizing computations to maximally take advantage of modern processors architectures, by avoiding pipeline stalls and cache misses. The most recent take on SaaT query evaluation is the technique known as JASS [37, 70], which has the additional advantage of being an *anytime* algorithm [82]: Since term contributions are considered in decreasing order of importance (that is, contributions to the final document score), query evaluation can terminate at any time to yield an approximate ranking.

Because of their fundamentally different approaches, SaaT methods make fewer assumptions about the underlying distribution of term weights, and thus their performance is less affected by transformer-generated representations compared to their DaaT counterparts. As we empirically demonstrate, this gives rise to differential (negative) performance artifacts, which is exactly the motivation that compels us to re-examine the performance of DaaT vs. SaaT methods in light of modern neural models.

Compact Storage and Retrieval. Although document-ordered and impact-ordered indexes represent vastly different organizations of the same data, there are additional optimizations that can apply to both. Index compression, for example, is an important inclusion in modern retrieval architectures, allowing more compact storage and better memory efficiency. A large body of prior work exists for compressing indexes; see, for example, the survey by Pibiri and Venturini [62]. Interestingly, prior work by Lin and Trotman [38] demonstrated that the latency of SaaT systems can be reduced with *uncompressed* indexes, but at a large space cost.

Another optimization that can reduce index size and accelerate retrieval is document identifier reassignment. The main idea is to reassign the integer identifiers of each document such that similar documents appear close together in the resulting index [15, 48, 67]. By clustering documents together, the delta gaps representing the document identifiers typically become smaller, thereby making compression more effective. Interestingly, document identifier reassignment has been shown to accelerate retrieval in both DaaT and SaaT systems, making it a useful pre-processing step during indexing [49].

Comparing Index Traversal Mechanisms. Although a wide range of work has focused on improving the efficiency of various search engines based on inverted indexes, there has been relatively little work on comparing different indexing and query processing approaches. One of the earliest studies was conducted by Fontoura et al. [19], who studied top-k retrieval strategies for in-memory document-ordered indexes. More recently, Mallia et al. [53] explored the interplay between compression schemes and retrieval latency with different DaaT algorithms. However, these studies did not examine impact-ordered indexes. Similarly, Mackenzie and Moffat [44] examined how different efficiency innovations interact with one another within the DaaT WAND family of dynamic pruning algorithms.

To our best knowledge, the most recent comparisons *across* index arrangements was the work of Crane et al. [9] and the work of Mackenzie et al. [49]. The experiments of Crane et al. [9] showed, on a number of standard IR test collections, that DaaT processing (via the BMW algorithm) was faster than SaaT retrieval (represented by the JASS system). Nevertheless, SaaT retrieval was shown to have a much better control of latency, exhibiting much lower *tail latency* than the DaaT techniques. For a relatively small fraction of queries, ill-behaved term weights render BMW much slower than JASS. Most recently, the work of Mackenzie et al. [49] employed JASS as a baseline against a new class of anytime DaaT retrieval algorithms. Their experiments showed that the novel anytime DaaT retrieval systems can outperform their SaaT counterparts, addressing some of the weaknesses previously attributed to DaaT algorithms.

This represents the state of affairs with respect the age-old DaaT vs. SaaT debate. However, as we will show, the advent of a new class of neural retrieval models means that a lot of these prior observations no longer hold.

2.2 Retrieval with Learned Representations

The performance benefits of the various optimization techniques described above depend naturally on the scoring function. For example, the ability to skip blocks of postings depends on the relationship between the various term weights and how they are distributed across the index (for block-based pruning approaches). Previous evaluations have been based on "traditional" scoring functions such as BM25 or language models [60], but recently the field has seen the emergence of models where term weights are *learned*; the community has been describing them as learned *sparse* representations [35, 36]. These models still rely on bag-of-words representations, thus top-k retrieval can still be captured by Eq. (1), and hence all the foregoing discussions about DaaT vs. SaaT approaches still apply. However, term weights are now supplied by neural networks (today, transformer-based pretrained language models), and learned in a supervised manner from large amounts of training data.

The first example of this class of models using transformers is DeepCT [12], which uses a regression model to learn term frequencies. Mackenzie et al. [47] showed that these frequencies affect the behavior of DaaT query evaluation approaches, although, with appropriate mitigation, it is possible to accelerate query evaluation without harming effectiveness. In other words, the distribution of term weights assigned by the neural model is qualitatively different from those based on an unsupervised model, say, BM25. Additional evidence for this finding comes from the work of Mallia et al. [55] in the context of their model called DeepImpact: query latency is noticeably longer with term weights assigned by a transformer model, compared to the BM25 score over the same sets of terms in each document. To further compound this issue, some learned models also assign weights to *queries*, thereby changing the relative importance between the terms and potentially hindering efficiency further. These interesting observations suggest that we should take another detailed look at query evaluation algorithms in the context of learned sparse retrieval models, and thus the virtues of DaaT vs. SaaT should be re-examined.

Note that our work only examines learned *sparse* representations for retrieval, where documents are represented by bags of words, i.e., the basis of the vector representation is the vocabulary space. There is, of course, another large class of learned *dense* representations, exemplified by models such as DPR [30] and ANCE [78]; see Lin et al. [42] for a survey. Although transformer-based models are involved in both classes of retrieval models, dense retrieval techniques require a completely different indexing and retrieval pipeline. For example, top-k retrieval is formulated as nearest neighbor search and implemented either with brute-force search or with approximate techniques such as HNSW [50]. These techniques do not use inverted indexes and thus our discussions of DaaT and SaaT query evaluation algorithms are not applicable. However, for the interested reader, a number of papers have attempted to draw conceptual connections between dense and sparse learned representations [35, 36, 41].

3 EXPERIMENTAL SETUP

Our experiments used the popular MS MARCO v1 passage corpus, comprising 8.8M passages, and all models were evaluated on the 6980 queries in the development set [4]. We also explored the v2 passage corpus in Section 6.4.

3.1 Retrieval Models

As points of comparison, we adopted the following baselines:

BM25 simply performs retrieval using the ubiquitous BM25 scoring function [65] over bag-of-words representations of the passages in the corpus. We set $k_1 = 0.82$ and $b = 0.68$, after tuning on a selection of training instances on the MS MARCO passage ranking test collection [40].

BM25 w/ doc2query–T5 [58, 59] (BM25-T5 for short) augments passages in the corpus with query predictions generated by the T5 [64] neural sequence-to-sequence model. The expanded passages are scored using BM25 at retrieval time, with the same BM25 formulation and parameters above. Thus, while neural models are involved in corpus preparation, the assignment of term weights does not involve any neural networks.

We examined the following retrieval models that leverage learned sparse representations using transformers:

DeepImpact [55] uses doc2query–T5 to identify dimensions in the passage's bag-of-words representation that should have non-zero weights (i.e., expansion terms) and learns a term weighting model based on a pairwise loss between relevant and non-relevant passages with respect to a query.

uniCOIL + doc2query–T5 [36] (uniCOIL-T5 for short) is a simplified variant of COIL [24] that assigns scalar weights to terms (as opposed to vector weights in the original COIL formulation). This model additionally benefits from doc2query–T5 expansions.

uniCOIL + TILDE [80] (uniCOIL-TILDE for short) can be best characterized as replacing the doc2query–T5 expansion component with an alternative model based on TILDE [81] that has lower inference costs but appears to be just as effective.

SPLADEv2 [20] represents an improvement over SPLADEv1 [21], which itself builds on SparTerm [3]. For this family of sparse retrieval models, the expansion component can be best characterized as being based on masked language modeling. SPLADEv2 further improves effectiveness via distillation techniques.

Note that for all models, reported experiments are based on our own code implementations and data preparation, with Anserini as a starting point (see details below). In all applicable cases, we started with checkpoints of the neural models provided by the authors. Thus, our results are quite close, but not exactly the same, as figures reported in the respective authors' original papers. For the learned sparse retrieval models, the corpus and queries are both pre-tokenized; the specific tokenization method depends on the model and how it was trained. Since these statistics may have an effect on efficiency, we provide a detailed analysis in Section 5.1. The prepared corpora already include term weights for document terms, and the same for the queries. Thus, none of these experiments involved neural inference, which eliminates a source of non-determinism in neural models.

3.2 Search Systems

We conducted experiments with three different retrieval systems:

Anserini [79] is an IR toolkit built on the open-source Lucene search library written in Java. The version of Anserini used in our experiments is based on Lucene 8.3.0, which uses BMW for query evaluation [25]. We made no special modifications to the default query evaluation settings other than applying the tuned BM25 parameters as discussed previously.

PISA [52] is an efficiency-focused IR system written in C++. In our experiments, PISA executes MaxScore [74] processing over SIMD-BP128 encoded postings lists [33]. While VBMW is currently the state-of-the-art approach for DaaT traversal, recent work has demonstrated that MaxScore is a better choice for larger values of k (such as $k = 1000$) and for long queries [53]. We also experiment with the *anytime* version of the PISA system described by Mackenzie et al. [49], which we discuss in more detail in Section 6.

JASS [37, 70] is, as far as we are aware, the only actively maintained open-source SaaT search engine today. It, too, was implemented in C++ for efficiency and uses a SIMD-enhanced Elias gamma encoding scheme for compressing the postings [71]. Query evaluation proceeds by processing impact-ordered postings using simple integer arithmetic and storing the results in an accumulator array managed with a unique page-table like algorithm [28]. JASS controls the tradeoff between query latency and effectiveness through a parameter ρ, which limits the total number of postings processed per query. Although the default implementation of JASS employs 16-bit accumulators (allowing maximum per document scores of $2^{16} - 1$), 32-bit accumulators were necessary in our initial experiments in order to avoid overflows, as the learned sparse impacts for document and query terms often result in scores exceeding $2^{16} - 1 = 65{,}535$. In Section 6, we present optimizations for supporting 16-bit and 8-bit accumulators.

For the learned sparse models, all implementations used the “pseudo-document” trick to assign custom impact scores to each document term. That is, if term X was assigned a (quantized) integer weight of ten by the transformer, we simply repeat the term ten times in a dynamically created “fake” document. This allows us to use all the existing systems without any modifications, by simply swapping in “sum of term frequency” as the scoring function, and has no impact on the indexing pipeline since we focus only on representing documents as bags-of-words. Note, however, that both DaaT systems *do not* pre-quantize the BM25 or BM25-T5 indexes, computing scores on the fly instead. In contrast, JASS relies on impact-based scoring, even for the BM25-based models.

3.3 Hardware and Evaluation

All experiments were conducted in memory using a single thread on a Linux machine with two 3.50 GHz Intel Xeon Gold 6144 CPUs and 512 GiB of RAM. Indexes were built with Anserini and exported to the Common Index File Format [39] before being imported into PISA and JASS—thus guaranteeing that each of the three systems has a semantically identical index. In addition, both PISA and JASS made use of document reordering via the *recursive graph bisection* algorithm [15, 48] which has been shown to improve compression and accelerate query processing for both DaaT and SaaT index traversal [49]. The latency measurements we report are the mean of three independent runs. Thus, we have adopted all current best practices to achieve the best possible results in a fair manner.

Output quality is measured in terms of mean reciprocal rank at cutoff 10 (RR@10), the official metric of the MS MARCO passage test collection; time is measured in terms of query latency in milliseconds when retrieving the top k documents; and space in terms of index size measured in megabytes. Note that latency figures do not include the time taken to encode or expand the queries for the learned sparse retrieval models since we used pre-tokenized queries with pre-computed weights. Besides, this time is simply a constant factor independent of the scoring technique.

Table 1. Experimental results for top $k = 1000$ retrieval on the development queries of the MS MARCO passage ranking test collection.

		Quality	Time	Space
Method		RR@10	Latency (ms)	Index Size (MB)
Anserini (Lucene): DaaT				
(1a)	BM25	0.187	40.1	661
(1b)	BM25-T5	0.277	62.8	1036
(1c)	DeepImpact	0.325	244.1	1417
(1d)	uniCOIL-T5	0.352	222.3	1313
(1e)	uniCOIL-TILDE	0.350	194.6	2067
(1f)	SPLADEv2	0.369	2140.0	4987
PISA: DaaT				
(2a)	BM25	0.187	8.3	739
(2b)	BM25-T5	0.276	11.9	1150
(2c)	DeepImpact	0.326	19.4	1564
(2d)	uniCOIL-T5	0.352	36.9	1358
(2e)	uniCOIL-TILDE	0.350	28.4	2108
(2f)	SPLADEv2	0.369	220.3	4326
JASS: SaaT				
Exact				
(3a)	BM25	0.187	15.8	1156
(3b)	BM25-T5	0.277	50.2	1452
(3c)	DeepImpact	0.326	39.3	2039
(3d)	uniCOIL-T5	0.352	147.2	1310
(3e)	uniCOIL-TILDE	0.350	83.5	1976
(3f)	SPLADEv2	0.369	314.0	3813
Approximate				
(4a)	BM25	0.186	12.4	1156
(4b)	BM25-T5	0.275	10.1	1452
(4c)	DeepImpact	0.319	12.6	2039
(4d)	uniCOIL-T5	0.338	14.9	1310
(4e)	uniCOIL-TILDE	0.338	15.4	1976
(4f)	SPLADEv2	0.324	15.3	3813

4 PRELIMINARY COMPARISON

Our first series of experiments compares the three retrieval systems (Anserini, PISA, and JASS) across different ranking models. The goal is to demonstrate that the term weight distributions derived from learned sparse models *do* cause performance issues with off-the-shelf DaaT and SaaT methods. Having thus identified the problems, we circle back in Section 6 to then address these issues.

Our main results are presented in Table 1, where each combination of retrieval model and system is characterized in terms of output quality, time, and space, with the top $k = 1000$ documents retrieved for each query. While PISA and JASS were designed specifically as platforms for research in query evaluation algorithms and thus engineered for speed, none of the systems had small index sizes explicitly as a design goal (beyond incidental effects on query evaluation performance that result from index compression techniques).

Each block of the table is devoted to a system, and rows represent each retrieval model. Note that JASS results occupy two blocks; the top one represents *exact* (i.e., rank-safe) processing (where all postings are processed) and the bottom one represents *approximate* processing. For the latter, we used a maximum of $\rho = 1$ million postings visited, based on heuristics provided by the JASS authors [37, 70].

There are a few takeaways from these results: First, our experiments confirm what previous researchers have already reported [47, 55]—that retrieval models based on learned sparse representations substantially alter the behavior of query evaluation algorithms (in a negative way). However, our results more thoroughly and comprehensively capture the effects, illustrating that these issues pervade learned sparse models more broadly. For DaaT approaches, both Anserini (Lucene) and PISA, we see that, in general, models with higher effectiveness are slower. In Lucene, the difference between BoW BM25 in row (1a) and the most effective model, SPLADEv2 in row (1f) is a whopping ∼50× slowdown. With PISA, the slowdown is "only" ∼25×, but nevertheless still quite dramatic. Even for models that are slightly less effective than SPLADEv2, for example, the uniCOIL models in rows (d) and (e), we observe substantially worse query evaluation performance, in both Lucene and PISA.

Second, it is clear that the query evaluation performance of Anserini (Lucene) is far behind that of PISA, both in absolute terms as well as relative degradation with the models that we have explored. Of course, this is not a fair comparison because Lucene is production-ready search library that is already widely deployed, whereas PISA is a research system. The performance deficiencies of Lucene have been discussed elsewhere [25] so we don't beleaguer the point further. For the remainder of this paper, we set aside Lucene and use PISA as the representative for DaaT retrieval.

To further examine different query evaluation algorithms, we applied WAND and BMW to SPLADEv2 in PISA in a separate experiment and found that these algorithms resulted in *slower* processing than an exhaustive ranked disjunction, with mean latency values of 619ms, 681ms, and 553ms, respectively. In our experiments, MaxScore vastly outperforms the WAND-based approaches; this result is due to MaxScore avoiding expensive sorting operations during query processing [53]. Essentially, WAND and BMW are "working hard" to compute which documents can be skipped, but the work is essentially wasted because in reality, few documents can actually be skipped if exact query evaluation is desired.

Third, the JASS results from Table 1 are largely consistent with what we already know about SaaT approaches from the literature. Exact query evaluation (i.e., exhaustively traversing all postings) with JASS is slower than PISA, but achieves comparable effectiveness. For the learned sparse retrieval models, although we do observe performance degradation with JASS as well, the slowdowns are slightly less than what we observed with PISA; "only" around 20× when comparing BoW BM25 (3a) with SPLADEv2 (3f).

The approximate query evaluation results with JASS are also consistent with previous findings. For "traditional" BM25 weighting, in rows (4a) and (4b), JASS is able to speed up query evaluation at the cost of small decreases in effectiveness. However, as the "model sophistication" increases, the heuristics provided by the JASS authors appear to result in an increased loss in effectiveness. Looking at rows (4d) to (4f), while JASS is much faster than PISA, it comes at a drop in effectiveness. Although the loss is less than 1% for BM25, and 4% on DeepImpact and uniCOIL, on SPLADEv2, the loss is over 12%, which can be attributed to the smaller fraction of total postings being processed [45]. Table 1 only captures two operating points for JASS, but we explore additional configurations in Section 6.3 that yield different effectiveness/efficiency tradeoffs.

Finally, in terms of index sizes, there's a general trend of larger index sizes as effectiveness increases; the underlying reasons will become obvious in the next section. However, for all models and systems, the space requirements are quite modest on the whole. In the context of modern

Table 2. Term statistics of documents and queries for different treatments of the MS MARCO passage corpus. Apart from the vocabulary size, the reported values are means computed across all documents/queries.

Method	$\lvert V\rvert$	Terms in Documents		Terms in Queries	
		Total	Unique	Total	Unique
BM25	2660824	39.8	30.1	4.5	4.4
BM25-T5	3929111	224.7	51.1	4.5	4.4
DeepImpact	3514102	4010.0	71.1	4.2	4.2
uniCOIL-T5	27678	5032.3	66.4	686.3	6.6
uniCOIL-TILDE	27646	8260.8	107.6	661.1	6.5
SPLADEv2	28131	10794.8	229.4	2037.8	25.0

servers, where RAM may be on the magnitude of hundreds of gigabytes, these differences are negligible. Nevertheless, deploying these models on much larger collections may result in the compact models being more attractive, especially as the index needs to be replicated across a distributed retrieval system.

5 INEFFICIENCIES OF LEARNED SPARSE MODELS

Based on the results of the previous experiments, it is clear that learned sparse models behave very differently from "traditional" scoring models such as BM25. In particular, although they can provide large effectiveness improvements, these come at the cost of query processing latency, irrespective of the underlying processing mode. Our next series of experiments examines this behavior and sheds light on *why* the learned sparse models are less efficient than their traditional counterparts.

5.1 Collection Statistics

As a starting point for this investigation, Table 2 shows descriptive statistics of the documents and queries across the different models. To this end, the total term count is best understood as the sum of all the weights assigned to all unique terms, either in the document or the query. On the other hand, the unique term count ignores the weighting component, and can be thought of as the number of postings entries associated with a document, or the number of postings lists traversed for a given query. The second column, $\lvert V\rvert$, denotes the vocabulary size, or the total number of unique terms in the collection; this is also the number of dimensions that are in the representation vectors of the documents and queries.

A few additional caveats are necessary for properly understanding these values. All statistics are computed based on our replication of the retrieval models, and thus may differ from figures reported in the authors' original papers due to differences in corpus preparation. In all cases, counts are computed by simply splitting text on whitespace. For BM25 and BM25-T5, these are *after* tokenization and stopword removal. For DeepImpact, uniCOIL and SPLADEv2, both documents and queries are pre-tokenized. Thus, there are qualitative differences, especially since uniCOIL and SPLADEv2 tokens are subwords derived from BERT.[1] Thus, for this reason, uniCOIL and SPLADEv2 have much smaller vocabulary sizes, but more unique terms per query.

Nevertheless, these statistics go a long way to explaining many of the observations from Table 1. It is clear that learned sparse retrieval models achieve higher effectiveness based on document expansion, and in some models, query term weighting and query expansion. Document expansion obviously yields longer documents (and bigger indexes), and together with query expansion, it is now clear why query evaluation efficiency degrades. For example, comparing DeepImpact and

[1]For example, "androgen receptor" is broken into "and ##rogen receptor", leading to a count of three tokens, not two.

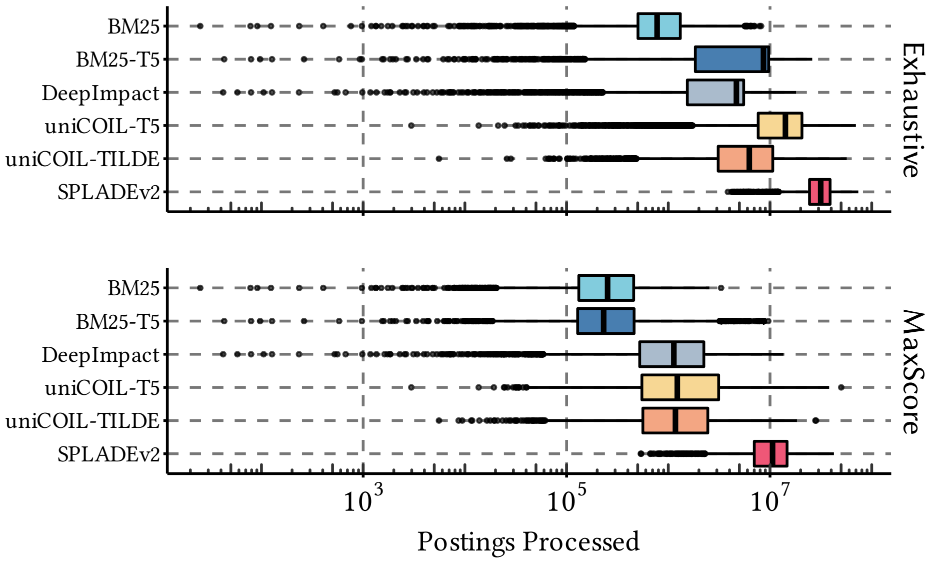

Fig. 2. The number of postings processed for each model with exhaustive retrieval (top) and for MaxScore processing with $k = 1000$ (bottom) on the MS MARCO v1 passage collection. More effective models typically require more postings to be traversed. Note the log x-axis.

uniCOIL-T5, we see that the latter performs query term weighting, whereas the former does not. This is likely the biggest source of query latency differences, row (2c) vs. row (2d) in Table 1; we explore this idea in the following subsection. Comparing the T5 and TILDE variants of uniCOIL, we observe that they achieve comparable effectiveness, but T5 is more compact (it has less document expansion). However, counter-intuitively, uniCOIL-TILDE, row (2e), is actually faster than uniCOIL-T5, row (2d). Figure 2 demonstrates why this is the case; even though the T5-based expansion is less aggressive, the TILDE-based index requires processing fewer postings across the MS MARCO queries, resulting in less total work. This observation also holds for MaxScore dynamic pruning, although the differences in pruning ability become less evident. Similar findings arise when examining the total number of documents scored during index traversal [57].

Finally, we note that SPLADEv2 likely derives its superior effectiveness from performing both query term weighting *and* query expansion, with new terms added to each query. Compared to the uniCOIL models, the SPLADEv2 queries contain around 4× more unique terms and the documents contain 2 – 4× more unique terms. We explore this in more detail in the next subsection.

5.2 Effects of Query Term Weighting and Query Expansion

Next, we conducted a small-scale ablation study on the uniCOIL and SPLADEv2 learned sparse models; these models perform query term weighting and query expansion. Understanding the effects of these different aspects on retrieval efficiency and effectiveness may shed more light on the possibility for optimization.

We ran both variants of the uniCOIL model with the DaaT PISA and SaaT JASS systems, using the same configurations as in Section 4, both with and without query term weighting. Table 3 shows the differences in efficiency and effectiveness that can be attributed to the weights assigned to the query terms by each uniCOIL model. Interestingly, it appears that query term weighting actually *helps* to accelerate the DaaT dynamic pruning methods. On the other hand, the effect on

Table 3. Efficiency and effectiveness for different configurations of both uniCOIL models using DaaT and (exact) SaaT retrieval. Interestingly, query term weighting makes retrieval faster for DaaT, whereas it has only a minor effect on the efficiency of SaaT.

Model	**Query Weighting**		**No Query Weighting**	
	RR@10	Latency	RR@10	Latency
PISA: DaaT				
uniCOIL-T5	0.352	36.9	0.312	92.8
uniCOIL-TILDE	0.350	28.4	0.309	77.3
JASS: SaaT				
uniCOIL-T5	0.352	147.2	0.317	152.2
uniCOIL-TILDE	0.350	83.5	0.309	93.2

Table 4. Efficiency and effectiveness for different configurations of SPLADEv2 using DaaT and (exact) SaaT retrieval. The default method, which employs both query term weighting and query expansion, is the most effective. Query term weighting accelerates retrieval, especially for DaaT, while query expansion is the largest source of added latency.

	Query Weighting		**No Query Weighting**	
	RR@10	Latency	RR@10	Latency
PISA: DaaT				
Query Expansion	0.369	220.3	0.281	430.6
No Query Expansion	0.338	30.8	0.317	46.9
JASS: SaaT				
Query Expansion	0.369	314.0	0.280	326.6
No Query Expansion	0.338	56.2	0.316	56.9

SaaT is marginal; the same number of postings are processed, with the only difference being the number of consequential heap insertions.

Similarly, we explored the effect of both query term weighting and query expansion on the SPLADEv2 model. Table 4 shows the results of this experiment. Again, we find that query term weighting is responsible for accelerating retrieval with the DaaT PISA system, but has only a minor impact on the SaaT JASS system. On the other hand, query expansion is the largest source of added latency to both retrieval systems. Since query expansion adds more postings lists to the computation, it is no surprise that it increases latency. It is interesting, however, that SPLADEv2 is capable of trading efficiency for effectiveness via query expansion. One possibility for future work is to train different levels of "expansion aggression" into SPLADEv2 to allow finer control of the resulting tradeoffs.

5.3 Qualitative Examination of Term Weights

The counter-intuitive nature of term weights assigned by learned sparse retrieval models is evident based on manual examination of model output. Building on the tokenization example above, for the query "androgen receptor define", the full SPLADEv2 query includes 25 unique tokens, representing an increase of 21 tokens beyond those in the original query, "and ##rogen receptor define". Figure 3 shows the entire query as ("term", weight) tuples.

Examining these tokens, the model assigns what appear to be reasonable weights to the original input terms: ("and", 225), ("##rogen", 251), ("receptor", 242), and ("define": 59). However, the large

('##rogen', 251) ('receptor', 242) ('and', 225) ('receptors', 189)
('hormone', 179) ('definition', 162) ('meaning', 99) ('genus', 89)
('is', 70) (',', 68) ('define', 59) ('the', 56) ('drug', 53) ('for', 46)
('ring', 38) ('gene', 37) ('are', 32) ('god', 25) ('what', 18) ('##rus', 15)
('purpose', 12) ('defined', 10) ('doing', 8) ('a', 4) ('goal', 4)

Fig. 3. The SPLADEv2 encoded query for the input "androgen receptor define". Blue terms appear in the original input query, orange terms correspond to different inflections of the original input terms, and pink terms are entirely new expanded terms. The terms are represented as ('term', weight) tuples, and are sorted descending on weight.

weight on "and" is a bit surprising, because the term "androgen" is actually represented by both "and" and "##rogen", but the first subword is conflated with a stopword. Many expansion terms that SPLADEv2 adds to the query do make sense, for example, ("hormone", 179), ("definition", 162), and ("meaning", 99). However, puzzling is the fact that the model also adds many stopwords to the query, including ("is", 70), ("the", 56), ("for", 46), and ("are", 32). Most non-sensical is the fact that "," (indeed, the comma) is added as an expansion term, with a relatively large weight of 68. Note that as far as we are aware, SPLADEv2 was trained without any special corpus processing, and punctuation is indeed part of BERT's vocabulary (and, as such, the inverted index vocabulary). In some cases, punctuation is semantically meaningful (e.g., apostrophes in names like O'Toole), but it is difficult to see the role that the comma might play in the context of this query. While it is difficult to argue with the overall effectiveness results (the high mean RR@10), the interpretability of these weights is not straightforward.

6 ACCELERATING RETRIEVAL

Our preliminary experiments have demonstrated that while learned sparse representations are much more effective than unsupervised weighting functions such as BM25, there is a substantial price to be paid in terms of larger index size and longer query evaluation latency. This observation holds for both DaaT and SaaT approaches, with the most expensive models requiring an order of magnitude more time than the BM25 baseline. In this section, we focus on techniques for accelerating query processing latency that aim to address these problems, and measure the resulting improvements.

6.1 Accelerating DaaT Traversal

One possibility for improving the efficiency of DaaT traversal is to employ an *approximate* processing technique, thereby allowing some effectiveness to be traded off for better efficiency. While DaaT systems cannot typically terminate early without large losses in effectiveness, recent work by Mackenzie et al. [49] introduced a simple yet effective approach for rapidly converging on the desired ranked list. Their *anytime* DaaT algorithm requires similar documents to be clustered together during indexing. Then, the entire inverted index is logically segmented into a series of *clusters*, where each cluster represents a set of similar documents, and term-wise upper-bound scores are stored *for each* index cluster. At query-time, a simple estimator provides an ordering over that set of clusters, which are visited greedily until either no new documents can enter the top-k or some other termination criteria are met; query processing *within* each cluster is the same as in a regular document-ordered index, but local cluster statistics provide tighter bounds for dynamic pruning. In contrast to a standard DaaT traversal, which works through the index from the lowest document identifier upwards, the anytime DaaT approach has the capacity to "jump" between clusters; Figure 4 provides an example of this compared to the typical DaaT algorithm.

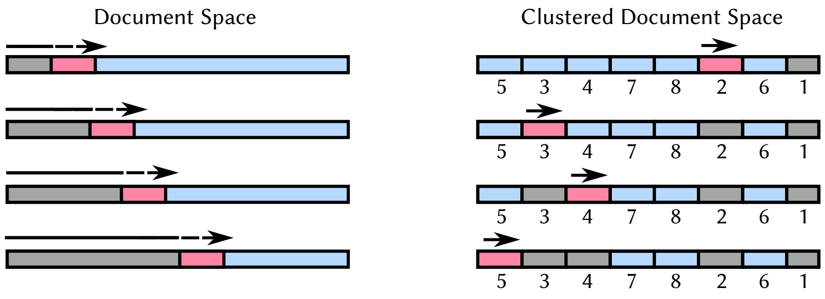

Fig. 4. A sketch comparing regular DaaT (left) vs. anytime DaaT (right) at four distinct points of the traversal. Gray sections represent documents that have been processed, whereas red sections represent the next documents to be processed. While DaaT moves through the document space from low-to-high document identifiers, anytime DaaT prioritizes the clusters of documents to process. By prioritizing the order in which clusters are visited, anytime DaaT can more rapidly converge on the final top-k results list.

We hypothesized that anytime DaaT can reduce the amount of processing required for learned sparse representations without large sacrifices in effectiveness. In our experiments, we assume that the underlying document reordering process acts as a pseudo-clustering algorithm. Then, we split the document space into 100 distinct (and uniformly-sized) clusters. Following Mackenzie et al., clusters are visited in decreasing order of the sum of the local term-wise upper-bound scores, and MaxScore processing is used within each cluster. We denote this system "A-PISA" (anytime PISA) in our experiments. While the idea of *anytime* DaaT is not novel, we are the first to apply such techniques to tackle the efficiency shortcomings of learned sparse representations.

Measuring the Improvements. Table 5 compares the default PISA approach with the A-PISA system using the same experimental setup as Section 4 (rank-safe processing, $k = 1000$). The top block of Table 5 is copied from the corresponding PISA results in Table 1 for ease of comparison. Regardless of the retrieval model, the anytime approach is always faster, achieving speedups in mean query evaluation latency between 1.3× to 1.8× with a negligible impact on effectiveness. Similar speedups were observed for other values of k. The only downside to the anytime DaaT extension is the additional space required. However, this overhead is less than 10% of the total index size, and we observe that the overhead decreases with more effective models.

6.2 Accelerating SaaT Traversal

Next, we turn our attention to accelerating SaaT traversal for learned sparse retrieval. Recall that the key idea with SaaT traversal is to visit segments of postings that are logically grouped together by score. By visiting the highest impact postings segments first, SaaT retrieval rapidly converges on the final top-k results, refining them as more postings are processed. We first propose two general improvements to JASS, and then propose a specific optimization targeted at reducing the negative effects of the term weights from the learned sparse representations (but can be used elsewhere as well). All three optimizations are novel and have not been described anywhere else.

Index Size Reduction. Previous papers, including Crane et al. [9], have compared DaaT and SaaT not only with respect to query evaluation latency, but also with respect to index size. They have reported SaaT indexes being approximately the same size as DaaT indexes. This is surprising, as an impact-ordered index should be storing fewer integers than a document-ordered index, since

Table 5. Comparing the default DaaT PISA approach to the anytime PISA approach with $k = 1000$.

Method	Quality RR@10	Time Latency (ms)	Space Index Size (MB)
PISA: Exact			
BM25	0.187	8.3	739
BM25-T5	0.276	11.9	1150
DeepImpact	0.326	19.4	1564
uniCOIL-T5	0.352	36.9	1358
uniCOIL-TILDE	0.350	28.4	2108
SPLADEv2	0.369	220.3	4326
Anytime PISA: Exact			
BM25	0.187	6.4	818
BM25-T5	0.276	8.1	1260
DeepImpact	0.327	11.2	1677
uniCOIL-T5	0.352	20.9	1376
uniCOIL-TILDE	0.350	19.3	2126
SPLADEv2	0.368	156.4	4346

SaaT stores the impact score once for a group of documents (a segment) whereas DaaT stores an impact score (or term frequency value) for each document.

Careful examination of the JASS indexes shows that indeed fewer integers are stored in the postings lists, but that there is an overhead to each segment of each postings list. This overhead includes storing pointers to the start of each segment, the segment's impact score, and so on. In the version of JASS we started with, these values were stored as uncompressed 32-bit and 64-bit integers, and in almost all cases, these integers were unnecessarily large for the values stored in the index. We addressed this issue by changing the encoding to variable-byte integers and removing unnecessary information from the segment headers.

Heap Score Caching. SaaT retrieval uses a min-heap to keep track of the top-k scoring documents via pointers into the global score accumulator table. The usual approach in SaaT is to check to see if the heap is full, and if not, to add the impact score to the accumulator before adding the accumulator to the heap. Only once the heap is full is a comparison performed to the bottom of the heap in order to maintain the min-heap property. This series of operations typically requires at least two comparisons per posting, as each individual scoring operation could result in a document being promoted into the top k.

Trotman and Crane [70] optimized this approach by re-arranging the order of operations. Instead of checking whether the heap is full or not, their approach adds the current impact score to the accumulator, which is then compared to the bottom of the heap (which is primed with zeros). If the score of the accumulator is greater than or equal to the score at the bottom of the heap, then the heap will need to be updated. This simple change reduces the typical number of comparisons per posting from two to one, and results in a 8% decrease in latency in their experiments. Our baseline version of JASS includes this optimization.

We observe, however, that this approach may result in unnecessary comparisons when score ties arise, a common situation with quantized impact scores. In particular, maintaining the min-heap ordering when ties are present requires on-the-fly tie breaking, which JASS achieves through a secondary comparison between internal document identifiers. Our additional novel micro-optimization is to simply cache the score at the bottom of the heap to avoid tie-breaking when a document cannot enter the heap.

A secondary benefit of this approach is that it allows a minimum heap entry score to be *primed* before querying, introducing the notion of top-k threshold estimation to SaaT search. Similar to DaaT retrieval, the retrieval remains "safe" so long as the initial estimation is lower than the final document score at rank k [54, 61]. If the estimation is too high, the heap will not fill, but the accumulator table will still be populated; a simple scan over the accumulator table is sufficient to recover the top-k scores.

In preliminary experiments, we found that applying oracle thresholds resulted in only modest speedups compared to simply allowing the threshold to grow organically as the heap fills, so we opted to use the latter approach in our experiments. Nevertheless, we believe this micro-optimization may be further exploited, and plan to examine it further in future work.

Query-Specific Impact Score Rescaling. In Section 3.2 we noted that working with learned sparse representations results in large accumulator values for many documents in many queries. These values are sufficiently large that they caused numeric overflow in the normally 16-bit accumulators used in JASS. To be clear, term weights are assigned by the learned sparse model (ultimately derived from neural inference) and we have no control over the range of the scores "out of the box".

The obvious solution is to increase the size of the accumulators to 32 bits. However, manipulating large quantities of 32-bit integers takes more time than manipulating the same quantity of 16-bit integers, due to three main reasons: First, 32-bit instructions can be expected to take longer to execute than the simpler 16-bit equivalents. Second, the CPU cache can hold twice as many 16-bit accumulators as 32-bit accumulators. Third, a single cache line read will read twice as many 16-bit accumulators as 32-bit accumulators. These last two points are particularly relevant when the document ids have been reordered.

An alternative to increasing the size of the accumulators is to decrease the accuracy of the impact score stored in the index. Crane et al. [8] examined the optimal size of the impact scores used in impact-ordered indexes and concluded that the best size to use (to increase precision while decreasing index size) was:

$$b = \left\lceil g + h \times 10^{-4} \times \sqrt{|d|} \right\rceil \tag{2}$$

where g and h are metric specific constants, $|d|$ is the number of documents in the collection, and b is the number of bits to use in the representation. It is worth emphasizing that Crane et al. examined *static*, index-wide scaling of the impact scores, which can be viewed as a single preprocessing optimization.

Unfortunately, the size of the impact scores in our learned sparse representations all fall within the bound suggested by Crane et al., making it impractical to further "squash" the score distribution without losing effectiveness. However, we note that the overflow of the accumulators is typically caused by the *combination* of query *and* document impact scores, not just the document scores alone.

To address this issue, we introduce a novel optimization we call *query-specific* impact score rescaling. Recall that SaaT query evaluation loads the postings list for each term, then sorts all segments from all terms from highest impact score to lowest impact score. We introduce a phase between these two in which we multiply the impact score for each segment by the query impact score. We then keep a running total of the highest score from each term. That score, M_q, is the largest possible score that any document can achieve for that query, and is given by:

$$M_q = \sum_{t \in q} \arg\max_d (I_{d,t} \times |t_q|) \tag{3}$$

where q is the query, t is a unique term from that query, $I_{d,t}$ is an impact score for document d containing t, and $|t_q|$ is the number of times t occurs in q.

Knowing this query-specific maximum score makes it possible to re-scale the scores to fit an arbitrary range. Without loss of generality, we use simple linear scaling:

$$I'_{d,t} = \frac{I_{d,t} \times |t_q|}{M_q} \times R \tag{4}$$

where $I'_{d,t}$ is the new term/document impact score, and R is the new maximum possible impact score. After processing, all documents will have scores between 0 and R. Obvious ranges to use are those that fit into a machine word easily manipulated by a modern CPU, i.e., 8-bit, 16-bit, or 32-bit. We explore these three settings in our experiments, where higher values are expected to yield better effectiveness at the cost of higher latency. It is worth emphasizing that this dynamic scaling is performed optionally and *per query*—that is, only when there is a potential for accumulator overflow do we rescale the score ranges; otherwise, query evaluation proceeds exactly as before.

Measuring the Improvements. The discussions above present three novel SaaT optimizations: two of these, index size reduction and heap caching, are *general* and can be applied beyond learned sparse retrieval models; the third optimization, dynamic impact score rescaling, directly targets the problems arising from the high query weights reported in Section 4, but can also be used beyond learned sparse retrieval.

The complete set of results is shown in Table 6: the top block is copied from the corresponding JASS results in Table 1 to facilitate easy comparison. The middle and bottom blocks show the results of the optimizations, with 16-bit accumulators and 8-bit accumulators, respectively. In terms of query latency, we observe large improvements across all models, particularly when dynamically rescaling from 32-bit to 16-bit accumulators (overall speedup of between 1.3× and 1.9×); moving to 8-bit accumulators yields additional gains, although not as dramatic (a speedup between 1.1× and 1.6× compared to the 16-bit rescaling). In terms of effectiveness, we observe no differences between 32-bit and 16-bit accumulators, although 8-bit accumulators can result in a small loss for some models. The new encoding scheme yields savings of between 5% and 40% of the total index size, with higher savings realized for collections with larger vocabularies.

Table 6 reports the cumulative gains of all our optimizations. However, we also conducted an ablation study to tease apart the effects of each individual improvement. We found that the new compression scheme resulted in a minor decrease in latency (on the order of 1–2 milliseconds). Similarly, heap caching resulted in a small but consistent improvements of around a 1.1× speedup in mean query latency. The remaining, and vast majority, of the latency reduction is directly attributable to dynamic impact score rescaling.

6.3 Effectiveness/Efficiency Tradeoffs

To recap thus far: Our preliminary experiments clearly identify the challenges that unusual term weight distributions from learned sparse retrieval models present for both DaaT and SaaT methods. We then proceeded to tackle these issues for both approaches. For DaaT, it appears that the application of a previously developed anytime technique provides an adequate solution. For SaaT, we developed three novel optimizations to address the challenges.

In our next set of experiments, we return to the age-old question of DaaT vs. SaaT, after both approaches have been optimized to handle learned sparse retrieval models. In order to explore

Table 6. Comparing the default SaaT JASS system with optimizations introduced in our work ($k = 1000$).

	Quality	Time	Space
Method	RR@10	Latency (ms)	Index Size (MB)
JASS: Default (32-bit)			
BM25	0.187	15.8	1156
BM25-T5	0.277	50.2	1452
DeepImpact	0.326	39.3	2039
uniCOIL-T5	0.352	147.2	1310
uniCOIL-TILDE	0.350	83.5	1976
SPLADEv2	0.369	314.0	3813
JASS + Optimizations (16-bit)			
BM25	0.187	8.2	729
BM25-T5	0.277	38.6	947
DeepImpact	0.326	23.7	1354
uniCOIL-T5	0.352	89.0	1139
uniCOIL-TILDE	0.350	54.2	1782
SPLADEv2	0.369	217.3	3595
JASS + Optimizations (8-bit)			
BM25	0.187	6.6	729
BM25-T5	0.270	24.6	947
DeepImpact	0.325	18.0	1354
uniCOIL-T5	0.351	71.5	1139
uniCOIL-TILDE	0.350	43.3	1782
SPLADEv2	0.364	194.9	3595

multiple points of the effectiveness/efficiency tradeoff space, we apply both exact and approximate versions of each approach. In particular, we run JASS with various settings of ρ, ranging from extremely aggressive early-termination (ρ = 100 thousand) to exact processing (ρ = 1 billion). Similarly, we vary the number of clusters to process during anytime DaaT traversal, R, with $R \in \{5, 10, 20, 50, 100\}$ out of a total of 100 clusters.

Figure 5 shows the resulting tradeoffs. Here, we report absolute measures of effectiveness (mean RR@10) as well as efficiency (mean query latency). This plot captures all combinations of retrieval models (as shapes) and systems (as colors). Furthermore, we show results for three values of $k = \{10, 100, 1000\}$ to cover different use cases, ranging from single-stage retrieval (smaller values of k) to candidate generation for reranking (larger values of k). Combinations that are on the Pareto-optimal frontier are highlighted, with non-optimal combinations shadowed. Similar trends were observed when measuring recall.

In this context, the Pareto-optimal frontier represents the best tradeoff that can be obtained for all explored combinations of retrieval models × systems. If a point lies on the frontier, it means that no other setting is able to achieve *both* higher effectiveness and lower mean query latency. There is no principled way to identify "better" or "worse" configurations along the frontier, as the selection of the operating point depends on the application scenario. Points along the frontier represent the best of "what's possible" for the system designer to choose from.

Our first observation is that, for $k = 10$ (the top panel), the anytime DaaT method is most frequently the best choice, with the converse holding true for $k = 1000$ (the bottom panel). These findings align with previous studies, which show that DaaT efficiency is much more sensitive to the value of k than SaaT retrieval [9]. Our second observation is that, quite amazingly, *all* models

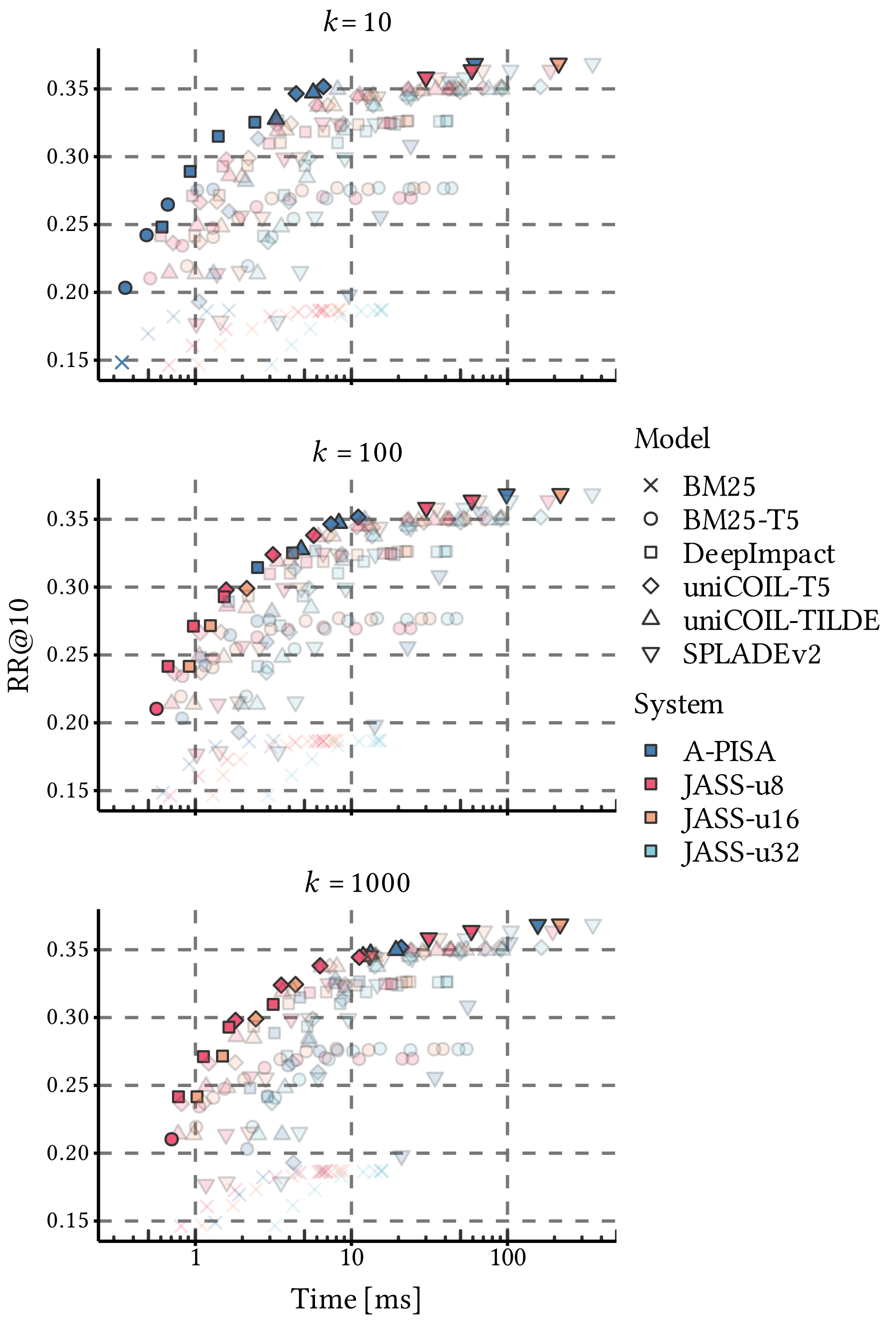

Fig. 5. Efficiency (mean query latency) vs. effectiveness (mean RR@10) for all configurations; those along the Pareto-optimal frontier are highlighted. Note that *every* retrieval model is Pareto-optimal under at least one combination of system and k.

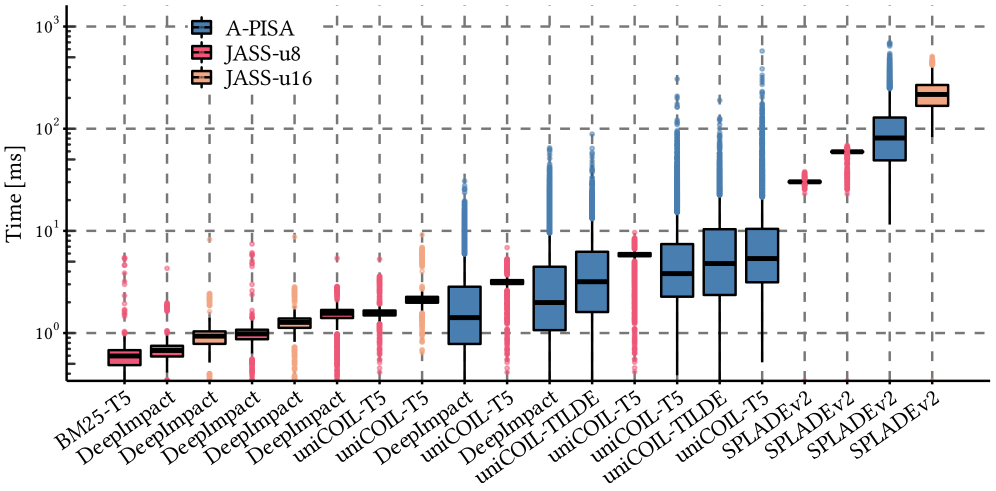

Fig. 6. Latency distribution of all Pareto-optimal configurations where $k = 100$ (that is, the highlighted points from the middle pane of Figure 5), sorted from lowest-to-highest effectiveness. All retrieval models appear on the frontier with the exception of BM25 (which is dominated by BM25-T5).

are Pareto-optimal in at least one system/k configuration. This means that there is no single system that dominates all others. Furthermore, A-PISA, JASS-u8, and JASS-u16 share points along the frontier (with JASS-u32 dominated in all cases). This means that depending on the desired tradeoffs between effectiveness and efficiency, the optimal choice varies across values of k, both in terms of the retrieval model as well as the query evaluation approach!

Finally, Figure 6 provides more details on query latency along the Pareto frontier for $k = 100$. The configurations have been arranged in increasing order of quality (x axis) with the corresponding latency distribution shown on the y axis as standard Tukey box-and-whiskers plots. It is important to note that the selection of the Pareto-optimal configurations was performed using *mean* query latency, and means might differ from medians or tail quantiles due to outliers.

Confirming prior work [9], we observe that while DAAT query evaluation *can* outperform SAAT query evaluation in terms of mean latency, better performance comes at the expense of less predictable latency, evidenced by the larger spread and higher percentile outliers in the A-PISA boxes. On the other hand, SAAT query evaluation, by design, yields much more predictable latency, as it enforces a strict limit on the total amount of allocated computation on a per query basis. It is also worth noting the tight clustering of query latency for the different JASS configurations with the SPLADEv2 model. Since SPLADEv2 aggressively expands queries (see Table 2), JASS almost always processes ρ postings, resulting in highly predictable performance. In contrast, for some of the other models, queries may have fewer than ρ postings to process, resulting in the lower tails (queries that are much faster), for example, in JASS with BM25-T5.

6.4 Scaling Up

So far, we have only experimented with the 8.8 million MS MARCO v1 passage collection. To determine whether the empirical findings hold on different collections, we ran a validation experiment on the much larger MS MARCO v2 passage collection.

Experimental Setup. The MS MARCO v2 passage collection consists of 138.4 million passages, representing more than a 15× scale increase compared to MS MARCO v1 [10]. Similar to the v1 collection, the v2 passages are drawn from English web documents. All models were evaluated on the 3903 queries in the first dev set. Collections were indexed in the same manner as the previous experiments with one exception: we increased the number of clusters from 100 to 2000 for the anytime DAAT mechanism. We explored one baseline and four learned sparse retrieval models that collectively represent the current state-of-the-art:

BM25 with default parameters $k_1 = 0.9$ and $b = 0.4$. For BM25 retrieval, we employ the official "augmented" version of the v2 corpus which expands each passage with the source document URL, title, and document headings [43].

SPLADE++SD and **SPLADE++ED** [22] represent some of the latest SPLADE models, which use MarginMSE knowledge distillation [26] with different hard negative sampling techniques to improve effectiveness. In particular, SPLADE++SD (self distillation) mines negative examples from a base SPLADE model before a subsequent training phase. In contrast, SPLADE++ED (ensemble distillation) uses a publicly available set of negative examples before further training. Both approaches use the coCondenser model [23] as their starting points.

ESPLADE-V and **ESPLADE-VI** [32] represent additional variants of the SPLADE family. These models incorporate a series of efficiency improvements (hence our notation, ESPLADE) over the original SPLADE family, with the aim of improving efficiency without greatly diminishing effectiveness. ESPLADE-V and ESPLADE-VI correspond to the (v): PLM with FLOPS and the (vi): BERT-Tiny models reported by Lassance and Clinchant [32].

Similar to the experimentation on MS MARCO v1, we apply both exact and approximate versions of both PISA and JASS on the MS MARCO v2 collection. In particular, we run JASS with various settings of ρ, ranging from extremely aggressive early-termination (ρ = 500 thousand) to exact processing (ρ = 1 billion). Similarly, we vary the number of clusters to process during anytime DaaT traversal, R, with $R \in \{100, 250, 500, 1000, 2000\}$ out of a total of 2000 clusters.

Effectiveness/Efficiency Tradeoffs. Figure 7 shows the resulting tradeoffs between the official RR@100 metric and mean query latency with $k = 1000$. As expected, we see similar trends as those on the much smaller v1 collection. There is a clear Pareto-optimal set of configurations which flows between different systems (including PISA and the different versions of JASS) as well as the different ranking models. However, we note that both SPLADE++ models are dominated by their ESPLADE counterparts. This is perhaps not surprising as the ESPLADE models are specifically tuned to provide good effectiveness/efficiency tradeoffs in the passage ranking context.

A few other interesting observations on the much larger collection: First, we can see the timings are around an order of magnitude larger on MS MARCO v2 than those on MS MARCO v1; nonetheless, highly effective configurations can still achieve top 1000 retrieval in the 100 to 200 millisecond range. Second, we find that the rank safe anytime DaaT mechanism (where $R = 2000$) is actually slower than the default out-of-the-box PISA system on the v2 collection. While anytime DaaT still provides useful points on the tradeoff curve (by allowing effectiveness to be traded for lower latency), this was quite surprising, and indicates that better document clustering and prioritization algorithms [49] may be necessary to achieve better performance. Finally, we observe similar tradeoffs for different values of k, again with the DaaT mechanisms becoming a better choice as k decreases. On the other hand, improvements to SaaT retrieval hold on the larger collection, with smaller accumulator values providing more efficient retrieval at the cost of minor effectiveness degradation.

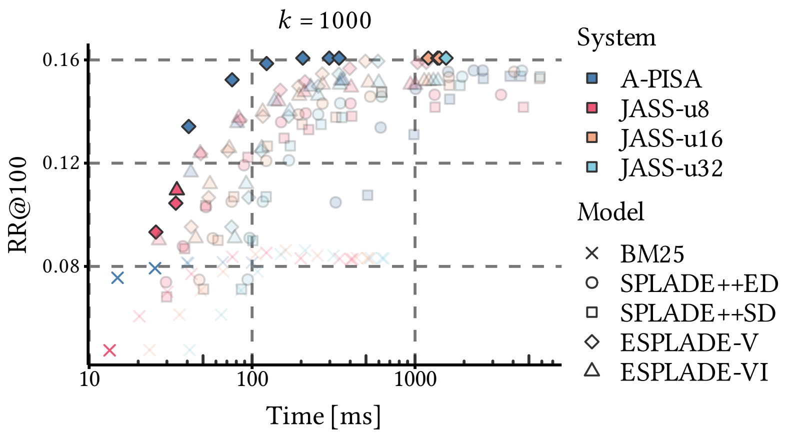

Fig. 7. Efficiency (mean query latency) vs. effectiveness (mean RR@100) for all configurations on MS MARCO v2; those along the Pareto-optimal frontier are highlighted.

It is also worth noting that the MS MARCO v2 collection contains substantially more documents than might typically be seen in a search engine shard in a production system. Trotman et al. [73], for example, suggest that eBay shards are "typically tens of millions of documents" and not the 138.4 million documents seen in v2. As such, the experiments in this section are on a collection that may be unrealistically large and take an unrealistically long time to search. Nevertheless, we believe the experiments are important as they provide a scalability point of reference even if not practical in a real-world application. Indeed, as collection processing becomes more complex, and as querying latency increases, we can expect shard sizes to decrease, unless, of course, CPU single core processing speed increases at a faster rate (which we do not believe is currently the case).

6.5 Experimental Summary

Coming back full circle to the age-old DAAT vs. SAAT debate, what have we learned? To begin, we demonstrated how classic DAAT and SAAT retrieval algorithms can be rendered less efficient by learned sparse models, and proposed new methods to combat these issues. With these "fixes", we find that DAAT algorithms remain a good choice when k is small (smaller than useful as a first-stage ranker, since rerankers typically process relatively large candidate sets), but suffer from difficult-to-predict tail latency. On the other hand, SAAT methods are very stable because the total workload is well managed, and they also scale well with increasingly large k values—making SAAT the best option as a first-stage ranker when used with learned sparse representations. However, our experiments paint a more complex picture, especially for "intermediate" values of k.

Before the advent of learned sparse retrieval models, the choices were relatively simple—the scoring model was largely fixed (e.g., BM25), and the practitioner selected between DAAT and SAAT, and then tuned a few parameters. Now, the choice of the model itself is an important consideration, because different sparse retrieval models dominate different parts of the Pareto-optimal frontier. For example, it is not possible to take an effective model like SPLADEv2 and "tune it" to operate in the high efficiency regime; it becomes necessary to switch to a completely different model.[2] Our study begins to clarify this complex design space, allowing a practitioner to map from operational considerations to good configuration choices.

[2]That said, it *is* possible to modify a model like SPLADEv2 to make it more efficient during training [32].

7 LIMITATIONS AND OPEN QUESTIONS

The rapidly evolving field of neural information retrieval provides a number of exciting research directions, and in this work we have focused on just one piece of the much larger puzzle. Our primary goal was to better understand the current landscape of latency, space, and effectiveness with different retrieval methods and models. While we believe that we have achieved this goal, we also wish to acknowledge some of the limitations of our work, and briefly discuss some open questions for future research.

Given the vast number of available choices of query processing systems, compression codecs, and implementation specific settings, we fixed a number of settings based on preliminary experimentation and guidance from the existing literature. For example, the choice of applying MaxScore processing as the representative method for top-k DaaT retrieval was based on Mallia et al. [53]. However, Mallia et al. used BM25 retrieval on traditional web corpora as their task. Although Mackenzie et al. [47] showed MaxScore processing to be on-par with VBMW using the DeepCT learned term frequencies on the MS MARCO v1 passage collection, it is not clear whether these findings will hold across different collections and ranking models. Furthermore, many alternative DaaT top-k retrieval algorithms have yet to be rigorously tested with learned sparse models, including those that use window-based postings lists, live-block mechanisms, and other optimizations [13, 16, 17, 31, 56, 66, 77]. Thus, a key open question is whether the assumptions derived from years of research on efficiency experimentation with traditional ranking models hold under newer learned sparse models.

Beyond fixing algorithmic settings and components, there is also the question of how efficiency and effectiveness can be traded off against one another through parameters such as k, the number of documents to retrieve. While it is well known that SaaT systems are not highly sensitive to the value of k, DaaT systems are known to be extremely sensitive to k, with smaller values allowing higher levels of dynamic pruning [9]. Hence, it is likely that there exists an even richer set of effectiveness/efficiency tradeoffs than those presented here. One promising research direction would be to revisit the problem of dynamic cutoff prediction (predicting the value of k on a per-query basis) in the context of learned sparse retrieval models [11, 34, 46].

Another limitation of our work is that we focus solely on the MS MARCO collections. While these collections have seen wide use by the community, it is important to also measure performance on different tasks and corpora. The BEIR meta-collection is one possible avenue for exploring learned sparse models in out-of-domain contexts [68], and has already seen attention from the community [32]. Another rapidly developing aspect of this line of research is on the models themselves; since writing this initial manuscript during late 2021 and through 2022, many newer models have been developed, including the efficient and distilled SPLADE models [22, 32] (used in Section 6.4) and the SpaDE model [7]. Keeping abreast of this rapidly changing research landscape represents another challenge for translating new findings into best practices.

Finally, our work only looked at learned sparse retrieval, where inverted indexes still reign as the go-to data structure for efficient top-k retrieval. Recently, however, a large body of research has focused on alternative representations beyond those readily supported by inverted indexes [35, 42]. A thorough comparison between the various options for supporting efficient top-k retrieval across all data structures, models, and data scales would benefit researchers and practitioners alike. There has already been some work in this direction [27], but more research is needed.

8 CONCLUSIONS AND FUTURE WORK

Retrieval models based on learned sparse representations are relatively new. Most work has focused on evaluating model effectiveness, but here we build on previous studies to examine effectiveness/efficiency tradeoffs. Our experiments demonstrate that the term weights generated by such models have substantive effects on the behavior of both DaaT and SaaT query processing algorithms. We then proceeded to address these issues, yielding large improvements to both DaaT and SaaT approaches. Finally, we performed a head-to-head comparison of DaaT and SaaT to arrive at an interesting finding: Which is better? *It depends*, as different sparse retrieval models in conjunction with different query evaluation approaches dominate different parts of the Pareto-optimal frontier. Thus, the "optimal" configuration will depend on operational considerations.

In future work, we hope to expand our experimental comparisons to different collections, including larger collections with longer documents and in out-of-domain settings. Finally, an obvious improvement for retrieval models based on learned sparse representations would be to incorporate efficiency considerations into the model training objective, much like the "learning to efficiently rank" thread of work from a decade ago [75, 76]. How to formulate an appropriate loss, however, is not obvious. Nevertheless, there are many exciting directions we are interested in pursuing in the near future.

SOFTWARE

Scripts and tools to reproduce our experiments can be found at the following URL: https://github.com/JMMackenzie/learned-sparse-shootout

ACKNOWLEDGEMENTS

This research was partially supported by the Australian Research Council Discovery Project DP200103136 and the Natural Sciences and Engineering Research Council (NSERC) of Canada. We thank Leonid Boytsov, Carlos Lassance, Antonio Mallia, Alistair Moffat, and Matthias Petri for helpful discussions related to this work. We also thank the anonymous referees who provided useful feedback to improve this manuscript.

REFERENCES

[1] G. Amati and C. J. V. Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems*, 20(4):357–389, 2002.

[2] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2001)*, pages 35–42, 2001.

[3] Y. Bai, X. Li, G. Wang, C. Zhang, L. Shang, J. Xu, Z. Wang, F. Wang, and Q. Liu. SparTerm: Learning term-based sparse representation for fast text retrieval. *arXiv:2010.00768*, 2020.

[4] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, M. Rosenberg, X. Song, A. Stoica, S. Tiwary, and T. Wang. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *arXiv:1611.09268v3*, 2018.

[5] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the Twelfth International Conference on Information & Knowledge Management (CIKM 2003)*, pages 426–434, 2003.

[6] K. Chakrabarti, S. Chaudhuri, and V. Ganti. Interval-based pruning for top-k processing over compressed lists. In *IEEE 27th International Conference on Data Engineering*, pages 709–720, 2011.

[7] E. Choi, S. Lee, M. Choi, H. Ko, Y.-I. Song, and J. Lee. SpaDE: Improving sparse representations using a dual document encoder for first-stage retrieval. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM 2022)*, pages 272–282, 2022.

[8] M. Crane, A. Trotman, and R. O'Keefe. Maintaining discriminatory power in quantized indexes. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management (CIKM 2013)*, pages 1221–1224, 2013.

[9] M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, and A. Trotman. A comparison of document-at-a-time and score-at-a-time query evaluation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM 2017)*, pages 201–210, 2017.

[10] N. Craswell, B. Mitra, E. Yilmaz, D. Campos, and J. Lin. Overview of the TREC 2021 deep learning track. In *Proceedings of the 30th Text REtrieval Conference (TREC 2021)*, 2021.

[11] J. S. Culpepper, C. L. A. Clarke, and J. Lin. Dynamic cutoff prediction in multi-stage retrieval systems. In *Proceedings of the 21st Australasian Document Computing Symposium (ADCS 2016)*, pages 17–24, 2016.

[12] Z. Dai and J. Callan. Context-aware sentence/passage term importance estimation for first stage retrieval. *arXiv:1910.10687*, 2019.

[13] C. M. Daoud, E. S. de Moura, D. Fernandes, A. S. da Silva, C. Rossi, and A. Carvalho. Waves: A fast multi-tier top-k query processing algorithm. *Information Retrieval Journal*, 20(3):292–316, 2017.

[14] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[15] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. Compressing graphs and indexes with recursive graph bisection. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016)*, pages 1535–1544, 2016.

[16] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. A candidate filtering mechanism for fast top-k query processing on modern CPUs. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2016)*, pages 723–732, 2013.

[17] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. Optimizing top-k document retrieval strategies for block-max indexes. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM 2013)*, pages 113–122, 2013.

[18] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2011)*, pages 993–1002, 2011.

[19] M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Zien. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proceedings of the VLDB Endowment*, 4(12):1213–1224, 2011.

[20] T. Formal, C. Lassance, B. Piwowarski, and S. Clinchant. SPLADE v2: Sparse lexical and expansion model for information retrieval. *arXiv:2109.10086*, 2021.

[21] T. Formal, B. Piwowarski, and S. Clinchant. SPLADE: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 2288–2292, 2021.

[22] T. Formal, C. Lassance, B. Piwowarski, and S. Clinchant. From distillation to hard negative sampling: Making sparse neural IR models more effective. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2022)*, pages 2353–2359, 2022.

[23] L. Gao and J. Callan. Unsupervised corpus aware language model pre-training for dense passage retrieval. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL 2022)*, pages 2843–2853, 2022.

[24] L. Gao, Z. Dai, T. Chen, Z. Fan, B. V. Durme, and J. Callan. Complementing lexical retrieval with semantic residual embedding. In *Proceedings of the 43rd European Conference on Information Retrieval (ECIR 2021), Part I*, pages 146–160, 2021.

[25] A. Grand, R. Muir, J. Ferenczi, and J. Lin. From MaxScore to Block-Max WAND: The story of how Lucene significantly improved query evaluation performance. In *Proceedings of the 42nd European Conference on Information Retrieval, Part II (ECIR 2020)*, pages 20–27, 2020.

[26] S. Hofstätter, S. Althammer, M. Schröder, M. Sertkan, and A. Hanbury. Improving efficient neural ranking models with cross-architecture knowledge distillation. *arXiv:2010.02666*, 2020.

[27] S. Hofstätter, N. Craswell, B. Mitra, H. Zamani, and A. Hanbury. Are we there yet? A decision framework for replacing term based retrieval with dense retrieval systems. *arXiv:2206.12993*, 2022.

[28] X.-F. Jia, A. Trotman, and R. O'Keefe. Efficient accumulator initialisation. In *Proceedings of the 15th Australasian Document Computing Symposium (ADCS 2010)*, 2010.

[29] C. Kamphuis, A. P. de Vries, L. Boytsov, and J. Lin. Which BM25 do you mean? A large-scale reproducibility study of scoring variants. In *Proceedings of the 42nd European Conference on Information Retrieval (ECIR 2020)*, pages 28–34, 2020.

[30] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, 2020.

[31] O. Khattab, M. Hammoud, and T. Elsayed. Finding the best of both worlds: Faster and more robust top-k document retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2020)*, pages 1031–1040, 2020.

[32] C. Lassance and S. Clinchant. An efficiency study for SPLADE models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2022)*, pages 2220–2226, 2022.

[33] D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. *Software: Practice and Experience*, 41(1):1–29, 2015.

[34] M. Li, X. Zhang, J. Xin, H. Zhang, and J. Lin. Certified error control of candidate set pruning for two-stage relevance ranking. *arXiv:2205.09638*, 2022.

[35] J. Lin. A proposed conceptual framework for a representational approach to information retrieval. *SIGIR Forum*, 55(2): 4.1–4.29, 2022.

[36] J. Lin and X. Ma. A few brief notes on DeepImpact, COIL, and a conceptual framework for information retrieval techniques. *arXiv:2106.14807*, 2021.

[37] J. Lin and A. Trotman. Anytime ranking for impact-ordered indexes. In *Proceedings of the ACM International Conference on the Theory of Information Retrieval (ICTIR 2015)*, pages 301–304, 2015.

[38] J. Lin and A. Trotman. The role of index compression in score-at-a-time query evaluation. *Information Retrieval Journal*, 20(3):199–220, 2017.

[39] J. Lin, J. Mackenzie, C. Kamphuis, C. Macdonald, A. Mallia, M. Siedlaczek, A. Trotman, and A. de Vries. Supporting interoperability between open-source search engines with the common index file format. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2020)*, pages 2149–2152, 2020.

[40] J. Lin, X. Ma, S.-C. Lin, J.-H. Yang, R. Pradeep, and R. Nogueira. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 2356–2362, 2021.

[41] J. Lin, X. Ma, J. Mackenzie, and A. Mallia. On the separation of logical and physical ranking models for text retrieval applications. In *Proceedings of the 2nd International Conference on Design of Experimental Search & Information REtrieval Systems (DESIRES 2021): CEUR Workshop Proceedings Vol-2950*, pages 176–178, 2021.

[42] J. Lin, R. Nogueira, and A. Yates. *Pretrained Transformers for Text Ranking: BERT and Beyond.* Morgan & Claypool Publishers, 2021.

[43] X. Ma, R. Pradeep, R. Nogueira, and J. Lin. Document expansions and learned sparse lexical representations for MSMARCO V1 and V2. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2022)*, 2022.

[44] J. Mackenzie and A. Moffat. Examining the additivity of top-k query processing innovations. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM 2020)*, pages 1085–1094, 2020.

[45] J. Mackenzie, F. Scholer, and J. S. Culpepper. Early termination heuristics for score-at-a-time index traversal. In *Proceedings of the 22nd Australasian Document Computing Symposium (ADCS 2017)*, pages 8.1–8.8, 2017.

[46] J. Mackenzie, J. S. Culpepper, R. Blanco, M. Crane, C. L. A. Clarke, and J. Lin. Query driven algorithm selection in early stage retrieval. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM 2018)*, pages 396–404, 2018.

[47] J. Mackenzie, Z. Dai, L. Gallagher, and J. Callan. Efficiency implications of term weighting for passage retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2020)*, pages 1821–1824, 2020.

[48] J. Mackenzie, M. Petri, and A. Moffat. Faster index reordering with bipartite graph partitioning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 1910–1914, 2021.

[49] J. Mackenzie, M. Petri, and A. Moffat. Anytime ranking on document-ordered indexes. *ACM Transactions on Information Systems*, 40(1):13.1–13.32, Jan. 2022.

[50] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

[51] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. Faster BlockMax WAND with variable-sized blocks. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2017)*, pages 625–634, 2017.

[52] A. Mallia, M. Siedlaczek, J. Mackenzie, and T. Suel. PISA: Performant indexes and search for academia. In *Proceedings of the Open-Source IR Replicability Challenge (OSIRRC 2019): CEUR Workshop Proceedings Vol-2409*, pages 50–56, 2019.

[53] A. Mallia, M. Siedlaczek, and T. Suel. An experimental study of index compression and DAAT query processing methods. In *Proceedings of the 41st European Conference on Information Retrieval (ECIR 2019)*, pages 353–368, 2019.

[54] A. Mallia, M. Siedlaczek, M. Sun, and T. Suel. A comparison of top-k threshold estimation techniques for disjunctive query processing. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM 2020)*, page 2141–2144, 2020.

[55] A. Mallia, O. Khattab, T. Suel, and N. Tonellotto. Learning passage impacts for inverted indexes. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 1723–1727, 2021.

[56] A. Mallia, M. Siedlaczek, and T. Suel. Fast disjunctive candidate generation using live block filtering. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining (WSDM 2021)*, pages 671–679, 2021.

[57] A. Mallia, J. Mackenzie, T. Suel, and N. Tonellotto. Faster learned sparse retrieval with guided traversal. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2022)*, pages 1901–1905, 2022.

[58] R. Nogueira and J. Lin. From doc2query to docTTTTTquery, 2019.

[59] R. Nogueira, W. Yang, J. Lin, and K. Cho. Document expansion by query prediction. *arXiv:1904.08375*, 2019.

[60] M. Petri, J. S. Culpepper, and A. Moffat. Exploring the magic of WAND. In *Proceedings of the 18th Australasian Document Computing Symposium (ADCS 2013)*, pages 58–65, 2013.

[61] M. Petri, A. Moffat, J. Mackenzie, J. S. Culpepper, and D. Beck. Accelerated query processing via similarity score prediction. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2019)*, pages 485–494, 2019.

[62] G. E. Pibiri and R. Venturini. Techniques for inverted index compression. *ACM Computing Surveys*, 53(6), 2020.

[63] J. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1998)*, pages 275–281, 1998.

[64] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[65] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[66] M. Siedlaczek, A. Mallia, and T. Suel. Using conjunctions for faster disjunctive top-k queries. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM 2022)*, pages 917–927, 2022.

[67] F. Silvestri. Sorting out the document identifier assignment problem. In *Proceedings of the 29th European Conference on Information Retrieval (ECIR 2007)*, pages 101–112, 2007.

[68] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems (NeurIPS 2021)*, 2021.

[69] N. Tonellotto, C. Macdonald, and I. Ounis. Efficient query processing for scalable web search. *Foundations and Trends in Information Retrieval*, 12(4–5):319–500, 2018.

[70] A. Trotman and M. Crane. Micro- and macro-optimizations of SaaT search. *Software: Practice and Experience*, 49(5): 942–950, 2019.

[71] A. Trotman and K. Lilly. Elias revisited: Group Elias SIMD coding. In *Proceedings of the 23rd Australasian Document Computing Symposium (ADCS 2018)*, pages 4.1–4.8, 2018.

[72] A. Trotman, A. Puurula, and B. Burgess. Improvements to BM25 and language models examined. In *Proceedings of the 19th Australasian Document Computing Symposium (ADCS 2014)*, pages 58–65, 2014.

[73] A. Trotman, J. Degenhardt, and S. Kallumadi. The architecture of eBay search. In *Proceedings of the SIGIR Workshop on eCommerce (eCom 2017): CEUR Workshop Proceedings Vol-2311*, pages 176–178, 2017.

[74] H. R. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Information Processing & Management*, 31(6): 831–850, 1995.

[75] L. Wang, J. Lin, and D. Metzler. Learning to efficiently rank. In *Proceedings of the 33rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2010)*, pages 138–145, 2010.

[76] L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2011)*, pages 105–114, 2011.

[77] Q. Wang, C. Dimopoulos, and T. Suel. Fast first-phase candidate generation for cascading rankers. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2016)*, pages 295–304, 2016.

[78] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. N. Bennett, J. Ahmed, and A. Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *Proceedings of the 9th International Conference on Learning Representations (ICLR 2021)*, 2021.

[79] P. Yang, H. Fang, and J. Lin. Anserini: Reproducible ranking baselines using Lucene. *Journal of Data and Information Quality*, 10(4):Article 16, 2018.

[80] S. Zhuang and G. Zuccon. Fast passage re-ranking with contextualized exact term matching and efficient passage expansion. *arXiv:2108.08513*, 2021.

[81] S. Zhuang and G. Zuccon. TILDE: Term independent likelihood moDEl for passage re-ranking. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 1483–1492, 2021.

[82] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

[83] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):6:1–6:56, 2006.

Received April 2022; Revised October 2022